

Практическое занятие №16

Наименование практического занятия: составление программ с использованием ООП.

Цели практического занятия: закрепить усвоенные знания, понятия, алгоритмы, основные принципы составления программ, приобрести навыки составления программ с ООП в IDE PyCharm Community.

Вариант №9

Постановка задачи:

Создайте класс "Калькулятор" с методами "сложение", "вычитание", "умножение" и "деление". Каждый метод должен принимать два аргумента и возвращать результат операции.

Код:

```
from colorama import init, Fore, Back, Style

print(Style.BRIGHT + Fore.GREEN)
print("+-----+ \n| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 | \n| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 | \n| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 | \n+-----+ \n")

print(Style.BRIGHT + Fore.YELLOW)
print(' '*7, "-" * 42)
print(' '*7, '| Выберите действие: ', " " * 19, "|")
print(' '*7, "-" * 42)
print(' '*7, '| 1. Сложение ', " " * 26, "|")
print(' '*7, "-" * 42)
print(' '*7, '| 2. Вычитание ', " " * 25, "|")
print(' '*7, "-" * 42)
print(' '*7, '| 3. Умножение ', " " * 25, "|")
print(' '*7, "-" * 42)
print(' '*7, '| 4. Деление ', " " * 27, "|")
print(' '*7, "-" * 42)
print(' '*7, '| 5. Возведение ', " " * 24, "|")
print(' '*7, "-" * 42)

# -----

math = int(input("Введите математическое выражение: A + B, A - B, A * B, A / B -> "))

a = int(input("Введите число A -> "))
b = int(input("Введите число B -> "))
```

```
class Calculator:
    def add(self, a, b):
        return(a + b)

    def sub(self, a, b):
        return(a - b)

    def mlp(self, a, b):
        return(a * b)

    def dvd(self, a, b):
        if b == 0:
            raise ValueError("На ноль делить нельзя, УЧИ МАТЕМАТИКУ !!!")
        else:
            return(a / b)

    def sqr(self, a, b):
        return(a ** b)

calc = Calculator()

if math == 1:
    result = calc.add(a,b)
elif math == 2:
    result = calc.sub(a,b)
elif math == 3:
    result = calc.mlp(a,b)
elif math == 4:
    result = calc.dvd(a,b)
elif math == 5:
    result = calc.sqr(a,b)

print(Style.BRIGHT + Fore.WHITE)
print("      Результат -> ", result)
print("  ")

# -----
```

Результат работы:



Протокол выполнения программы:

1. Импортируются необходимые модули и функции из библиотеки `colorama` для оформления текста в консоли.
2. Выводится меню с перечнем доступных действий: сложение, вычитание, умножение, деление и возведение в степень.
3. Пользователь вводит номер выбранного действия.
4. Пользователь вводит значения переменных A и B.
5. Определяется класс `Calculator` с методами для выполнения арифметических операций.
6. Создается экземпляр класса `Calculator`.
7. Выполняется выбранная арифметическая операция в зависимости от номера, введенного пользователем.
8. Результат выводится в консоль.
9. Программа завершается.

Постановка задачи №2:

Создание базового класса "Работник" и его наследование для создания классов "Менеджер" и "Инженер". В классе "Работник" будут общие методы, такие как "работать" и "получать зарплату", а классы-наследники будут иметь свои уникальные методы и свойства, такие как "управлять командой" и "проектировать системы".

Код:

```
class Worker:
    def __init__(self, **kwargs):
        self.attributes = kwargs

    def work(self):
        print("Я работаю")

    def receive_salary(self):
        print("Урааа, я получил зарплату")

class Manager(Worker):
    def manage_a_team(self):
        print("Я управляю командой")

class Engineer(Worker):
    def design_systems(self):
        print("Я проектирую системы")

manager = Manager(name='Петр', position='Менеджер', salary=100000,
department='Отдел продаж')
engineer = Engineer(name='Андрей', position='Инженер', salary=80000,
specialization='Системный аналитик')

print(manager.attributes)
print(engineer.attributes)

# worker.work()
# worker.receive_salary()
# manager.work()
# manager.receive_salary()
# manager.manage_a_team()
# engineer.work()
# engineer.receive_salary()
# engineer.design_systems()
```

Протокол выполнения программы:

1. Определение базового класса "Работник" с методами `__init__`, `work` и `receive_salary`. Метод `__init__` принимает произвольное количество аргументов в формате ключ-значение и сохраняет их в словаре `attributes`.
2. Определение класса "Менеджер", наследующегося от класса "Работник" с добавлением метода `manage_a_team`.
3. Определение класса "Инженер", наследующегося от класса "Работник" с добавлением метода `design_systems`.

4. Создание объекта `manager` класса "Менеджер" с передачей аргументов `name='Петр'`, `position='Менеджер'`, `salary=100000`, `department='Отдел продаж'`.
5. Создание объекта `engineer` класса "Инженер" с передачей аргументов `name='Андрей'`, `position='Инженер'`, `salary=80000`, `specialization='Системный аналитик'`.
6. Вывод атрибутов объектов `manager` и `engineer` с помощью `print(manager.attributes)` и `print(engineer.attributes)`.

Результат выполнения программы:

```
олия\Десктоп\ГЗ\ГЗ10\pz10_pz.py
{'name': 'Петр', 'position': 'Менеджер', 'salary': 100000, 'department': 'Отдел продаж'}
{'name': 'Андрей', 'position': 'Инженер', 'salary': 80000, 'specialization': 'Системный аналитик'}
PS C:\Users\Анатолий\Desktop\PZ>
```

Постановка задачи №3:

Для задачи из блока 1 создать две функции, `save_def` и `load_def`, которые позволяют сохранять информацию из экземпляров класса (3 шт.) в файл и загружать ее обратно. Использовать модуль `pickle` для сериализации и десериализации объектов Python в бинарном формате.

Код программы:

```
import pickle

class MyClass:
    def __init__(self, attribute1, attribute2):
        self.attribute1 = attribute1
        self.attribute2 = attribute2

def save_def(obj, filename):
    with open(filename, 'wb') as f:
        pickle.dump(obj, f)

def load_def(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)

obj1 = MyClass(1, 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.')
obj2 = MyClass(2, 'Sed euismod, nisl vitae tincidunt lacinia, nisi nisi ullamcorper nisi, in aliquet massa nisl eget metus.')
obj3 = MyClass(3, 'Proin euismod, augue nec aliquet malesuada, nisl velit egestas velit, eget euismod libero odio vitae lectus')

save_def(obj1, 'obj1.pkl')
save_def(obj2, 'obj2.pkl')
```

```
save_def(obj3, 'obj3.pkl')

loaded_obj1 = load_def('obj1.pkl')
loaded_obj2 = load_def('obj2.pkl')
loaded_obj3 = load_def('obj3.pkl')

print(loaded_obj1.attribute1, loaded_obj1.attribute2)
print(loaded_obj2.attribute1, loaded_obj2.attribute2)
print(loaded_obj3.attribute1, loaded_obj3.attribute2)
```

Протокол выполнения программы:

1. Импортируется модуль `pickle`.
2. Определяется класс `MyClass` с двумя атрибутами: `attribute1` и `attribute2`.
3. Определяются функции `save_def` и `load_def` для сохранения и загрузки объектов класса `MyClass` в файл.
4. Создаются три объекта класса `MyClass` с разными значениями атрибутов.
5. Объекты сохраняются в файлы с помощью функции `save_def`.
6. Объекты загружаются из файлов с помощью функции `load_def`.
7. Значения атрибутов загруженных объектов выводятся на экран.

Вывод:

Я закрепил усвоенные знания, понятия, алгоритмы, основные принципы составления программ, приобрел навыки составления программ с ООП в IDE PyCharm Community.